**Code For JWT Token And Security Layer:**

**Entity : User**

package com.app.entities;

import java.util.Collection;

import java.util.HashSet;

import java.util.Set;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

import javax.persistence.ManyToMany;

import org.springframework.security.core.GrantedAuthority;

import org.springframework.security.core.authority.SimpleGrantedAuthority;

import org.springframework.security.core.userdetails.UserDetails;

@Entity

public class User implements UserDetails {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String username;

private String email;

private String address;

private String password;

private String phoneNo;

@ManyToMany // This is an example; adjust mapping based on your schema

private Set<Role> roles = new HashSet<>();

// Constructor for a regular user

public User(String username, String email, String address, String password, String phoneNo, Set<Role> roles) {

this.username = username;

this.email = email;

this.address = address;

this.password = password;

this.phoneNo = phoneNo;

this.roles = roles;

}

// Constructor for the admin

public User() {

this.username = "admin";

this.email = "admin@example.com";

this.address = "Admin Address";

this.password = "adminpassword";

this.phoneNo = "0000000000";

this.roles = new HashSet<>();

// Example of setting admin role; ensure Role class and its instances exist

this.roles.add(new Role("ROLE\_ADMIN"));

}

// Getters and Setters

public Long getId() {

return id;

}

public void setId(Long id) {

this.id = id;

}

public String getUsername() {

return username;

}

public void setUsername(String username) {

this.username = username;

}

public String getEmail() {

return email;

}

public void setEmail(String email) {

this.email = email;

}

public String getAddress() {

return address;

}

public void setAddress(String address) {

this.address = address;

}

public String getPassword() {

return password;

}

public void setPassword(String password) {

this.password = password;

}

public String getPhoneNo() {

return phoneNo;

}

public void setPhoneNo(String phoneNo) {

this.phoneNo = phoneNo;

}

public Set<Role> getRoles() {

return roles;

}

public void setRoles(Set<Role> roles) {

this.roles = roles;

}

@Override

public Collection<? extends GrantedAuthority> getAuthorities() {

Set<GrantedAuthority> authorities = new HashSet<>();

for (Role role : roles) {

authorities.add(new SimpleGrantedAuthority(role.getName()));

}

return authorities;

}

@Override

public boolean isAccountNonExpired() {

return true;

}

@Override

public boolean isAccountNonLocked() {

return true;

}

@Override

public boolean isCredentialsNonExpired() {

return true;

}

@Override

public boolean isEnabled() {

return true;

}

@Override

public String toString() {

return "User{" +

"id=" + id +

", username='" + username + '\'' +

", email='" + email + '\'' +

", address='" + address + '\'' +

", password='" + password + '\'' +

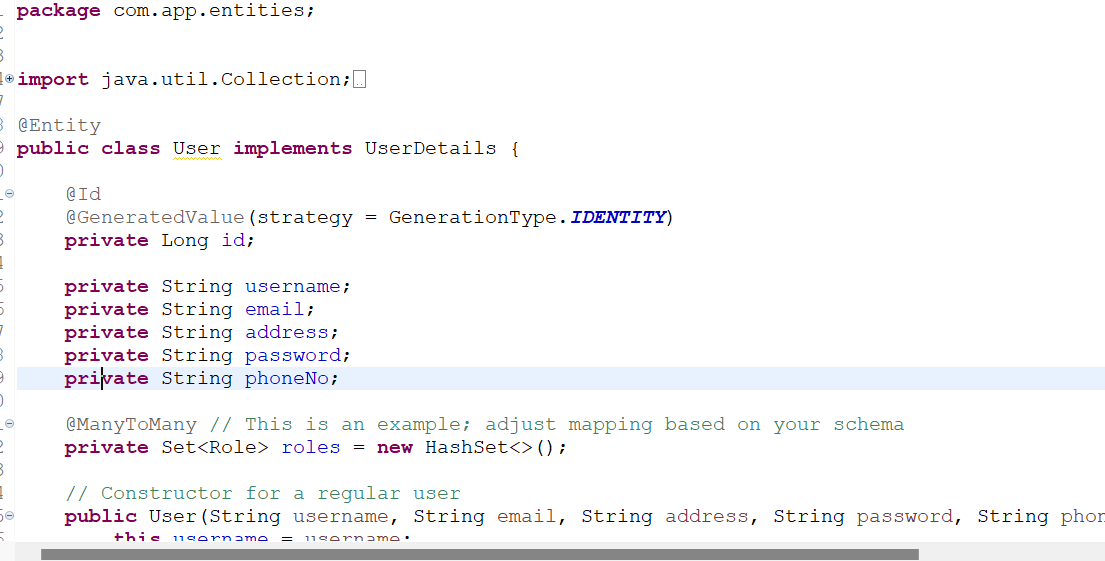
", phoneNo='" + phoneNo + '\'' +

", roles=" + roles +

'}';

}

}



**Dao-Layer:**

package com.app.dao;

import org.springframework.data.jpa.repository.JpaRepository;

import org.springframework.stereotype.Repository;

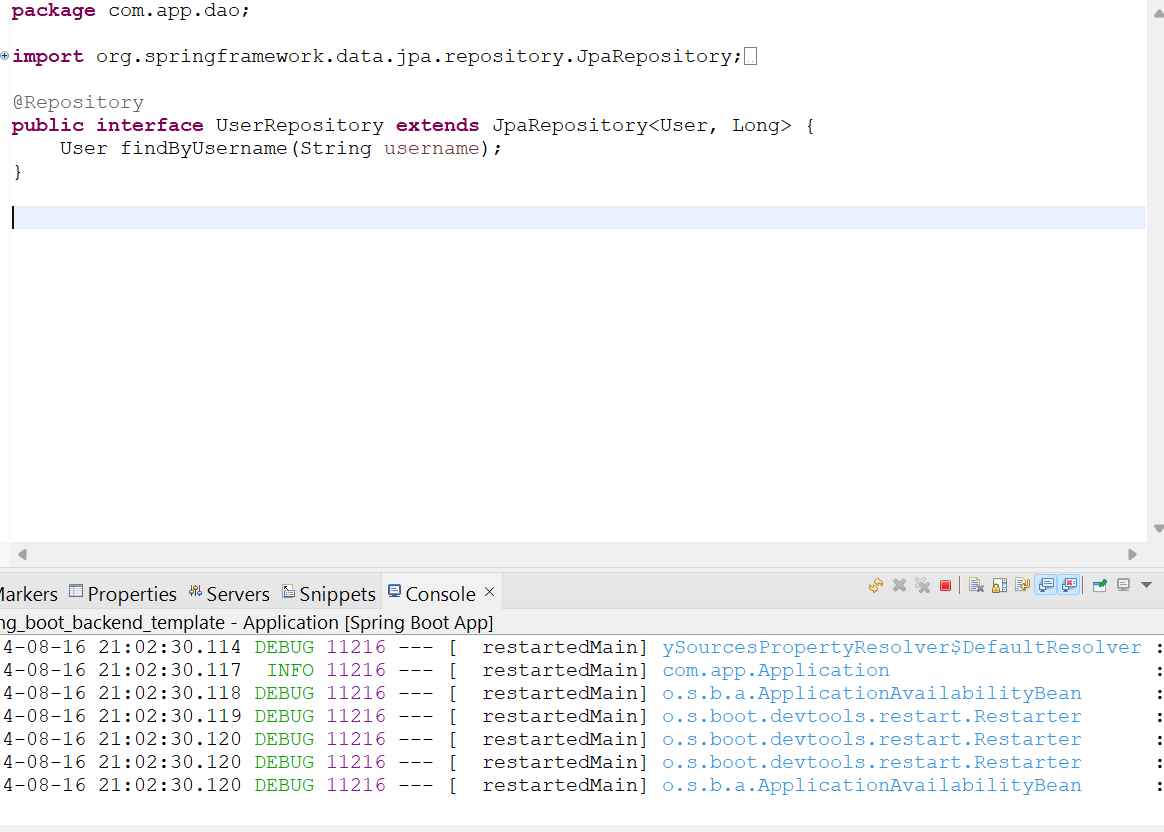
import com.app.entities.User;

@Repository

public interface UserRepository extends JpaRepository<User, Long> {

User findByUsername(String username);

}



**SecurityLayer:**

JWT Util:

package com.app.security;

import io.jsonwebtoken.Claims;

import io.jsonwebtoken.Jwts;

import io.jsonwebtoken.SignatureAlgorithm;

import org.springframework.stereotype.Component;

import java.util.Date;

import java.util.function.Function;

@Component

public class JwtUtil {

private String SECRET\_KEY = "mysecretkey";

public String extractUsername(String token) {

return extractClaim(token, Claims::getSubject);

}

public Date extractExpiration(String token) {

return extractClaim(token, Claims::getExpiration);

}

public <T> T extractClaim(String token, Function<Claims, T> claimsResolver) {

final Claims claims = extractAllClaims(token);

return claimsResolver.apply(claims);

}

private Claims extractAllClaims(String token) {

return Jwts.parser().setSigningKey(SECRET\_KEY).parseClaimsJws(token).getBody();

}

private Boolean isTokenExpired(String token) {

return extractExpiration(token).before(new Date());

}

public String generateToken(String username) {

return Jwts.builder()

.setSubject(username)

.setIssuedAt(new Date(System.currentTimeMillis()))

.setExpiration(new Date(System.currentTimeMillis() + 1000 \* 60 \* 60 \* 10)) // 10 hours

.signWith(SignatureAlgorithm.HS256, SECRET\_KEY)

.compact();

}

public Boolean validateToken(String token, String username) {

final String extractedUsername = extractUsername(token);

return (extractedUsername.equals(username) && !isTokenExpired(token));

}

}

**=================================================================================**

**Services:**

package com.app.services;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import com.app.dao.UserRepository;

import com.app.entities.User;

import com.app.entities.Role;

import org.springframework.security.core.userdetails.UserDetails;

import org.springframework.security.core.userdetails.UserDetailsService;

import org.springframework.security.core.userdetails.UsernameNotFoundException;

import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

import org.springframework.security.core.authority.SimpleGrantedAuthority;

import java.util.List;

import java.util.Optional;

import java.util.Set;

import java.util.stream.Collectors;

import java.util.Collections;

@Service

public class UserService implements UserDetailsService {

@Autowired

private UserRepository userRepository;

private BCryptPasswordEncoder passwordEncoder = new BCryptPasswordEncoder();

public User saveUser(User user) {

user.setPassword(passwordEncoder.encode(user.getPassword()));

return userRepository.save(user);

}

public List<User> getAllUsers() {

return userRepository.findAll();

}

public Optional<User> getUserById(Long id) {

return userRepository.findById(id);

}

public User getUserByUsername(String username) {

return userRepository.findByUsername(username);

}

public void deleteUser(Long id) {

userRepository.deleteById(id);

}

public boolean authenticateUser(String username, String password) {

User user = userRepository.findByUsername(username);

return user != null && passwordEncoder.matches(password, user.getPassword());

}

@Override

public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {

User user = userRepository.findByUsername(username);

if (user == null) {

throw new UsernameNotFoundException("User not found");

}

// Convert roles to SimpleGrantedAuthority

Set<SimpleGrantedAuthority> authorities = user.getRoles().stream()

.map(role -> new SimpleGrantedAuthority(role.getName())) // Assuming Role has a getName() method

.collect(Collectors.toSet());

return new org.springframework.security.core.userdetails.User(user.getUsername(), user.getPassword(), authorities);

}

}

**RequestFilter:**

package com.app.security;

import com.app.services.UserService;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.security.core.userdetails.UserDetails;

import org.springframework.security.core.userdetails.UsernameNotFoundException;

import org.springframework.stereotype.Component;

import org.springframework.web.filter.OncePerRequestFilter;

import javax.servlet.FilterChain;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import java.io.IOException;

import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;

import org.springframework.security.core.context.SecurityContextHolder;

import org.springframework.security.web.authentication.WebAuthenticationDetailsSource;

@Component

public class JwtRequestFilter extends OncePerRequestFilter {

@Autowired

private JwtUtil jwtUtil;

@Autowired

private UserService userService;

@Override

protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain chain)

throws ServletException, IOException {

final String authorizationHeader = request.getHeader("Authorization");

String username = null;

String jwt = null;

if (authorizationHeader != null && authorizationHeader.startsWith("Bearer ")) {

jwt = authorizationHeader.substring(7);

username = jwtUtil.extractUsername(jwt);

}

if (username != null && SecurityContextHolder.getContext().getAuthentication() == null) {

UserDetails userDetails = this.userService.loadUserByUsername(username);

if (jwtUtil.validateToken(jwt, userDetails.getUsername())) {

UsernamePasswordAuthenticationToken usernamePasswordAuthenticationToken =

new UsernamePasswordAuthenticationToken(userDetails, null, userDetails.getAuthorities());

usernamePasswordAuthenticationToken

.setDetails(new WebAuthenticationDetailsSource().buildDetails(request));

SecurityContextHolder.getContext().setAuthentication(usernamePasswordAuthenticationToken);

}

}

chain.doFilter(request, response);

}

}

**================================================================================**

Security-

package com.app.security;

import com.app.services.UserService;

import javax.sql.DataSource;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.security.authentication.AuthenticationManager;

import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;

import org.springframework.security.config.annotation.web.builders.HttpSecurity;

import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;

import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;

import org.springframework.security.config.http.SessionCreationPolicy;

import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

import org.springframework.security.crypto.password.PasswordEncoder;

import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;

import org.springframework.security.config.annotation.method.configuration.EnableGlobalMethodSecurity;

import org.springframework.security.config.annotation.web.builders.HttpSecurity;

import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;

import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;

import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

import org.springframework.security.crypto.password.PasswordEncoder;

@Configuration

@EnableWebSecurity

public class SecurityConfig extends WebSecurityConfigurerAdapter {

@Autowired

private UserService userService;

@Autowired

private DataSource dataSource;

@Autowired

private JwtRequestFilter jwtRequestFilter;

@Override

protected void configure(AuthenticationManagerBuilder auth) throws Exception {

auth.userDetailsService(userService).passwordEncoder(passwordEncoder());

auth.jdbcAuthentication()

.dataSource(dataSource)

.usersByUsernameQuery("SELECT username, password, enabled FROM users WHERE username = ?")

.authoritiesByUsernameQuery("SELECT username, role FROM user\_roles WHERE username = ?")

.passwordEncoder(passwordEncoder());

}

@Override

@Bean

**public** AuthenticationManager authenticationManagerBean() **throws** Exception {

**return** **super**.authenticationManagerBean();

}

@Bean

**public** PasswordEncoder passwordEncoder() {

**return** **new** BCryptPasswordEncoder();

}

}

**================================================================================**

**Controller:**

package com.app.controller;

import com.app.entities.User;

import com.app.security.JwtUtil;

import com.app.services.UserService;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.http.ResponseEntity;

import org.springframework.security.authentication.AuthenticationManager;

import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;

import org.springframework.security.core.AuthenticationException;

import org.springframework.web.bind.annotation.\*;

@RestController

@RequestMapping("/auth")

public class AuthController {

@Autowired

private AuthenticationManager authenticationManager;

@Autowired

private JwtUtil jwtUtil;

@Autowired

private UserService userService;

@PostMapping("/signin")

public ResponseEntity<?> createAuthenticationToken(@RequestBody User user) throws Exception {

try {

authenticationManager.authenticate(

new UsernamePasswordAuthenticationToken(user.getUsername(), user.getPassword())

);

} catch (AuthenticationException e) {

throw new Exception("Incorrect username or password", e);

}

final String jwt = jwtUtil.generateToken(user.getUsername());

return ResponseEntity.ok(jwt);

}

}

**AuthController:**

import com.app.entities.User;

import com.app.security.JwtUtil;

import com.app.services.UserService;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.http.ResponseEntity;

import org.springframework.security.authentication.AuthenticationManager;

import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;

import org.springframework.security.core.AuthenticationException;

import org.springframework.web.bind.annotation.\*;

@RestController

@RequestMapping("/auth")

public class AuthController {

@Autowired

private AuthenticationManager authenticationManager;

@Autowired

private JwtUtil jwtUtil;

@Autowired

private UserService userService;

@PostMapping("/signin")

public ResponseEntity<?> createAuthenticationToken(@RequestBody User user) throws Exception {

try {

authenticationManager.authenticate(

new UsernamePasswordAuthenticationToken(user.getUsername(), user.getPassword())

);

} catch (AuthenticationException e) {

throw new Exception("Incorrect username or password", e);

}

final String jwt = jwtUtil.generateToken(user.getUsername());

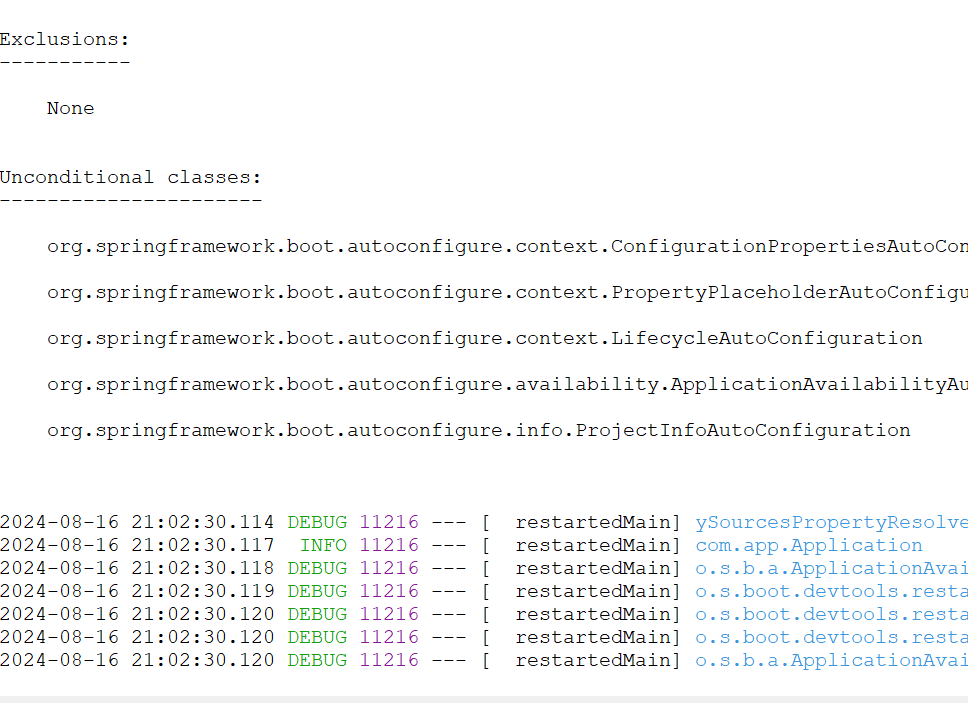
return ResponseEntity.ok(jwt);

}

}

**================================================================================**

***Port Started:7070***



***Fetch Database:***

